SSCM Exercise 5

Nikolaus Czernin - 11721138

```
library("tidyverse")
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(knitr)

# Custom print function
print_ <- function(...) print(paste(...))

set.seed(11721138)
```

Task 1

```
x1 <- c(-0.673, -0.584, 0.572, -0.341, -0.218, 0.603, -0.415, -0.013, 0.763, 0.804, 0.054, 1.746, -0.47
x2 <- c(0.913, -0.639, 2.99, -5.004, 3.118, 0.1, 1.128, 0.579, 0.32, -0.488, -0.994, -0.212, 0.413, 1.4

data.frame(
  Sample=c("x1", "x2"),
  Mean = c(mean(x1), mean(x2))
) %>% kable()
```

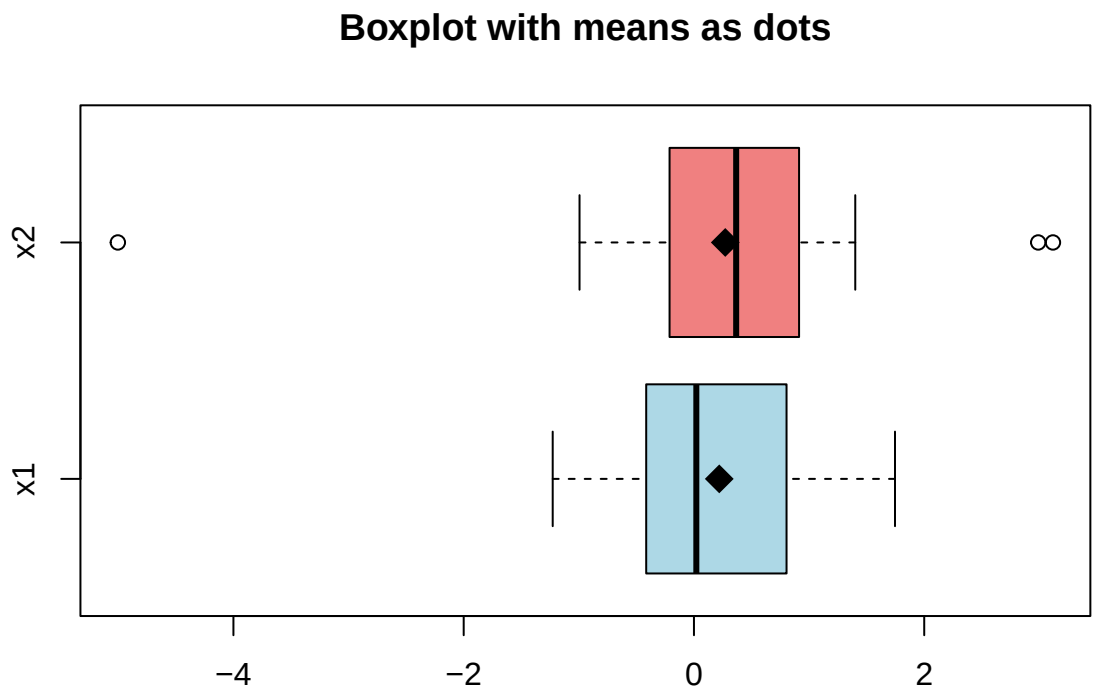
Sample	Mean
x1	0.2198182
x2	0.2717222

Plotting

```r
# Means
mean_x1 <- mean(x1)
mean_x2 <- mean(x2)

boxplot(x1, x2,
        horizontal = TRUE,
        names = c("x1", "x2"),
        main = "Boxplot with means as dots",
        col = c("lightblue", "lightcoral")
        )

points(mean_x1, 1, cex=2 , pch = 18)
points(mean_x2, 2 , cex=2, pch = 18)
```



Looking at the boxplots, the means seem to be pretty much the same almost. The first sample has fewer outliers though, a wider outer quantile distance and is more right-skewed.

Bootstrapping

```r
bootstrap.t.indep <- function(data.1, data.2, m=10000, ...) {
  # compute the original test-statistic
  original.t <- t.test(x1, x2)$statistic
  # generate bootstrap samples
  bs.samples.1 <- lapply(1:m, function(x) sample(data.1, replace = TRUE))
```

```
  bs.samples.2 <- lapply(1:m, function(x) sample(data.2, replace = TRUE))
  # compute the estimated parameters of the samples
  bs.ts <- sapply(1:m, function(i) t.test(bs.samples.1[[i]], bs.samples.2[[i]])$statistic)
  # we now have a student's t distribution
  # we can extract the p-value from it
  p.value = mean(abs(bs.ts) >= abs(original.t))
  # also get the confidence intervals, i.e. the quantiles of the t-stats
  ci.95 <- quantile(bs.ts, c(0.05, 0.95))
  ci.99 <- quantile(bs.ts, c(0.01, 0.99))
  # return whatever was created here
  list(p.value = p.value, ci.95 = ci.95, ci.99 = ci.99)
}

bootstrap.t.indep(x1, x2, m=30)
```

```
## $p.value
## [1] 0.9333333
##
## $ci.95
##        5%       95%
## -2.362550  1.354239
##
## $ci.99
##        1%       99%
## -2.547657  1.714015
```

Here I define a function for non-parametric bootstrapping as a test. The function first computes the estimator, in this case the student's t-test on the original two samples. It then iterates m times and each time applys non-parametric bootstrapping by sampling from the original samples with replacement to create 2 new samples, which are compared again with t tests. The t statistics of these m tests form a sort of student's t distribution, for wich I calculate a p value and confidence intervals.

```
bootstrap.t.centered <- function(data.1, data.2, m=10000) {
  # compute the mean of the combined samples
  mean.center <- mean(c(data.1, data.2))
  print("Centering the means by" %>% paste(-mean.center %>% round(3)))
  # center the 2 samples
  data.1 <- data.1 - mean.center
  data.2 <- data.2 - mean.center
  # apply the independent bootstrap esimation
  bootstrap.t.indep(data.1, data.2, m=m)

  }

bootstrap.t.centered(x1, x2, m=30)
```

```
## [1] "Centering the means by -0.243"
```

```
## $p.value
## [1] 0.9666667
##
## $ci.95
```

```
##        5%       95%
## -1.777542  1.273930
##
## $ci.99
##        1%       99%
## -2.098267  1.573056
```

The function here works similarly to the one before, it only applies combined mean centering before generating samples and applying the estimators beforehand.

Applying the functions on t-tests

```
results.bootstrap.t.indep <- bootstrap.t.indep(x1, x2)
results.bootstrap.t.centered <- bootstrap.t.centered(x1, x2)
```

```
## [1] "Centering the means by -0.243"
```

```
res <- data.frame(
  Independed.Bootstrap.Sampling=c(
    results.bootstrap.t.indep$ci.95 %>% round(4) %>% paste(collapse = " - "),
    results.bootstrap.t.indep$ci.99 %>% round(4) %>% paste(collapse = " - "),
    results.bootstrap.t.indep$p.value
  ),
  Mean.Centered.Bootstrap.Sampling=c(
    results.bootstrap.t.centered$ci.95 %>% round(4) %>% paste(collapse = " - "),
    results.bootstrap.t.centered$ci.99 %>% round(4) %>% paste(collapse = " - "),
    results.bootstrap.t.centered$p.value
  )
)

rownames(res) <- c("95% Conf.Level", "99% Conf.Level", "p-Value")
res %>%
  kable(caption="Comparison of Bootstrap Sampling methods on t-tests")
```

Table 2: Comparison of Bootstrap Sampling methods on t-tests

	Independed.Bootstrap.Sampling	Mean.Centered.Bootstrap.Sampling
95% Conf.Level	-2.1541 - 1.3704	-2.1464 - 1.3706
99% Conf.Level	-2.9426 - 1.9066	-2.8924 - 1.9422
p-Value	0.9083	0.912

Im getting p-values well over 0.9, so we are far from being able to reject the null-hypothesis of the samples having different means.

Permutation

```r
permutation_test <- function(data.1, data.2, m=4){
  # get the t statistic of the original samples
  t.original <- t.test(data.1, data.2)$statistic
  # combine the datasets
  data.combined <- c(data.1, data.2)
  # create bootstrap samples from the combined data
  t.bs <- sapply(1:m, function(x) {
    # create a permutation of the combined dataset
    permutation_idx <- sample(1:length(data.combined), size = length(data.1), replace = FALSE)
    # get 2 samples at the sizes of the original samples
    permutation.1 <- data.combined[permutation_idx]
    permutation.2 <- data.combined[-permutation_idx]
    # perform a t test and return the t statistic
    t.test(permutation.1, permutation.2)$statistic
  })

  # calculate the p value
  p.value <- mean(abs(t.bs) >= abs(t.original))
  # also get the confidence intervals, i.e. the quantiles of the t-stats
  ci.95 <- quantile(t.bs, c(0.05, 0.95))
  ci.99 <- quantile(t.bs, c(0.01, 0.99))
  # return whatever was created here
  list(p.value = p.value, ci.95 = ci.95, ci.99 = ci.99)

}



results.permutation_test <- permutation_test(x1, x2, m=10000)

res <- data.frame(
  "Independed Bootstrap Sampling"=c(
    results.permutation_test$ci.95 %>% round(4) %>% paste(collapse = " - "),
    results.permutation_test$ci.99 %>% round(4) %>% paste(collapse = " - "),
    results.permutation_test$p.value
  )
)

rownames(res) <- c("95% Conf.Level", "99% Conf.Level", "p-Value")
res %>%
  kable(caption="Permutation Sampling method on t-tests")
```

Table 3: Permutation Sampling method on t-tests

	Independed.Bootstrap.Sampling
95% Conf.Level	-1.6752 - 1.6483
99% Conf.Level	-2.2779 - 2.2709
p-Value	0.9175

Instead of adapting the first test function I made, I implemented a new function. It again creates a function that iterates m times and applies a t-test on random index permutations of the combined dataset. Using

the same formula as before I calculate the p values and the confidence intervals on the t statistics of the m students t test statistics.

The p value of the test is again ~0.9, so we are not rejecting the Null-hypothesis of the two samples having different means.

T-Test and Wilcoxon test

```
t.test(x1, x2)
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = -0.11881, df = 23.027, p-value = 0.9065
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.9556081  0.8518000
## sample estimates:
## mean of x mean of y
## 0.2198182 0.2717222
```

```
wilcox.test(x1, x2)
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  x1 and x2
## W = 181, p-value = 0.6572
## alternative hypothesis: true location shift is not equal to 0
```

The results of the base R `t.test()` also has a p-value of ~0.9, which is in line with my simulated tests and my conclusion of keeping the null-hypothesis.

The p-value of the Mann-Whitney test is ~0.66, which is also high enough for us not to be able to reject the null-hypothesis.

Task 2

```
n <- 200

# define function to sample the used variables from their distributions
x1 <- rnorm(n, mean = 2, sd = sqrt(3))
x2 <- runif(n, 2, 4)
x3 <- runif(n, -2, 2)
e <- rt(n, df = 5)

# define the function y
# Model: y = 3 + 2*x1 + x2 + e
```

```r
y <- 3 + 2 * x1 + x2 + e

# Generate a sample of size n
data <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)

data %>% head()
```

```
##          y        x1       x2          x3
## 1 6.661351 1.4019237 3.290549  0.77121233
## 2 7.127742 0.7400803 2.137931 -0.70210518
## 3 9.782905 0.7489008 3.777093  1.14821945
## 4 7.504739 0.6525578 2.334418 -0.86125898
## 5 7.181872 0.2496512 2.714391  0.38804432
## 6 8.129926 0.7756917 3.532724  0.06168864
```

I define functions that source the variables from their respective distributions and can be passed to the y function to create the model output. I use piping to create a dataframe of size 200 with all parameters and the output variable y.

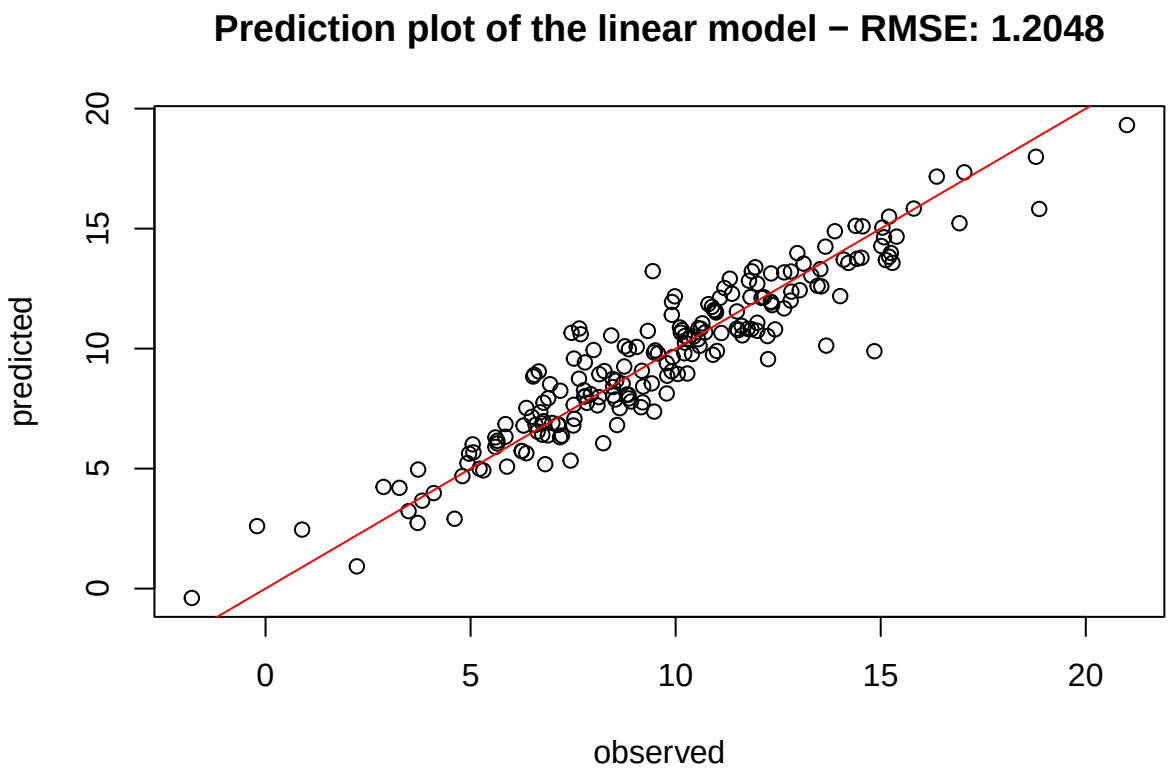
Residual bootstrap

```r
# custom rmse function
get_rmse <- function(model, r=4) sqrt(mean(model$residuals^2)) %>% round(r)

# fitting the linear model
lin <- lm(y ~ x1 + x2 + x3, data=data)
lin %>% summary()
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7861 -0.6979  0.0291  0.7476  4.9549
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.71552    0.45759   8.120 5.10e-14 ***
## x1           1.98722    0.05197  38.236  < 2e-16 ***
## x2           0.77007    0.14770   5.214 4.68e-07 ***
## x3           0.01592    0.07233   0.220    0.826
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.217 on 196 degrees of freedom
## Multiple R-squared:  0.8842, Adjusted R-squared:  0.8825
## F-statistic: 499.1 on 3 and 196 DF,  p-value: < 2.2e-16
```

```r
yhat <- predict(lin, data %>% select(-y))
plot(
  data$y,
  yhat,
  main="Prediction plot of the linear model - RMSE:" %>% paste(., get_rmse(lin)),
  ylab="predicted", xlab="observed"
)
abline(coef = c(0,1), col="red")
```



In the linear model, all included variables are significant, even x3, which played no part in creating the output variable and essentially distorts the data, but its coefficient is also the smallest.
Just visually, the fit is fine. I also report the RMSE in the heading.

Residual Bootstrap Regression

```r
# get the residuals
residuals <- lin$residuals
# create the bootstrap sample

residual.bootstrap.coeffs <- function(data, yhat, residuals, m=1000){
    # generate new residuals via bootstrapping, m times
    residuals.resampled <- sapply(1:m, function(j) sample(residuals, replace = TRUE))
    # we now get a matrix (n, m), where n is the size of the original sample
    # and m is the number of bootstraps were doing, e.g. 1000
```

```
    # create new predictions by adding the sampled residuals to the yhat
    # do this by adding the columns of residuals.resampled to the yhat
    yhat.bootstrapped <- apply(residuals.resampled, 2, function(rs) rs + yhat)
    # now we have another (n, m) matrix
    # fit a new model to each of the bootstrapped predictions,
    # without using e (epsilon)
    new.models <- lapply(1:m, function(j) lm(yhat.bootstrapped[,j] ~ x1 + x2 + x3, data=data))
    # we now have m different model objects and now we extract their coefficients
    # return the coefficients of the bootstrapped model
    coeffs <- lapply(new.models, coef) %>%
      do.call(rbind, .)
    # we now have a matrix of size (m, 4), where the 4 are the coefficients incl intercept
    coeffs
    # now extract the quantiles for each of the 4 coefficients
    apply(coeffs, 2, function(b) quantile(b, probs=c(0.05, 0.95)))


}

residual.bootstrap.coeffs(data, yhat, residuals)
```

```
##     (Intercept)       x1        x2         x3
## 5%     2.949199 1.900626 0.5235686 -0.1006502
## 95%    4.460691 2.066832 1.0009130  0.1451359
```

In the chunk above I define a test that takes the residuals of a linear model and creates m bootstrap samples by sampling with replacement from them. These resampled residuals are then added as not-so-random noise to the predicted response of the former linear model. By then trying to again fit m different models to these adjusted y values, we are simulating a correction of the models to the y values corrected by the residuals, i.e. nudging them around in the range of the actually observed values.
The m different models then have each 4 coefficients, and each of them has m different versions, for which I then grab the confidence intervals.

Since the confidence interval of the x3 variable is centered around 0, I presume that it is not really necessary for the model, even though it was found as significant by the `lm()` function.

Pairs Bootstrap Regression

```
# create the bootstrap sample
pairs.bootstrap.coeffs <- function(data, m=1000){
    bootstrap.samples <- lapply(1:m, function(j) data[sample(1:nrow(data), replace = TRUE),])
    # print(bootstrap.samples %>% length())
    # print(bootstrap.samples[[2]])
    # we now got m different resampled dataframes
    # now refit new models with them
    new.models <- lapply(bootstrap.samples, function(d) lm(y ~ x1 + x2 + x3, data=d))
    # we now have m models
    # now extract their coefficients and turn them into a matrix
    coeffs <- lapply(new.models, coef) %>%
      do.call(rbind, .)
    # now extract the quantiles for each of the 4 coefficients
```

```
    apply(coeffs, 2, function(b) quantile(b, probs=c(0.05, 0.95)))


}

pairs.bootstrap.coeffs(data)
```

```
##     (Intercept)       x1        x2         x3
## 5%     2.987873 1.909424 0.5304636 -0.1117285
## 95%    4.432463 2.070098 1.0101433  0.1465699
```

In this chunk I define a test that resamples from the original dataset, instead of the residuals. It samples rows of the dataset with replacement, exactly as many rows as the original dataset had. Then it fits the new data in a new linear model. All that is repeated m times and then coefficients' distributions' confidence intervals are returned.
In this case, the confidence interval of the variable x3 is no longer surrounding 0, so if I didnt know any better, I'd presume that this time around that the true model likely even includes this variable. Other than that the confidence intervals are very close to what they have been in the residual approach.

Approach comparison

The first approach sampled from the results of an initial model fitting, whereas the second sampled the original data. I there assume, that the first approach makes some kind of assumption that the model was already doing something right, like for example variable selection. The second approach on the other hand has never even seen any model before.
The first approach, while bootstrap sampling always uses the same predictors and input data, but always to predict something slightly different, whereas the second always tries to predict the same output using different permutations of its original input everytime.

Summary

Bootstrapping is a method where we create many new samples from our data by either resampling the data directly (non-parametric) or generating samples based on an assumed model (parametric). We use these samples to calculte confidence intervals, test statistics, or model estimtes. Parametric bootstrap assumes a specific distribution, which can give more precise results if the assumption is correct but may give misleading results if the model is wrong. Non-parametric bootstrap is flexible and works well for any dataset, but it may require more computational effort and does not smooth out sampling noise. Bootstrapping is powerful because it relies on the data itself to understand variability and does not need complex formulas for confidence bounds or tests.